CPEN 311: Digital Systems Design

Metastability and Synchronization

Learning Objectives

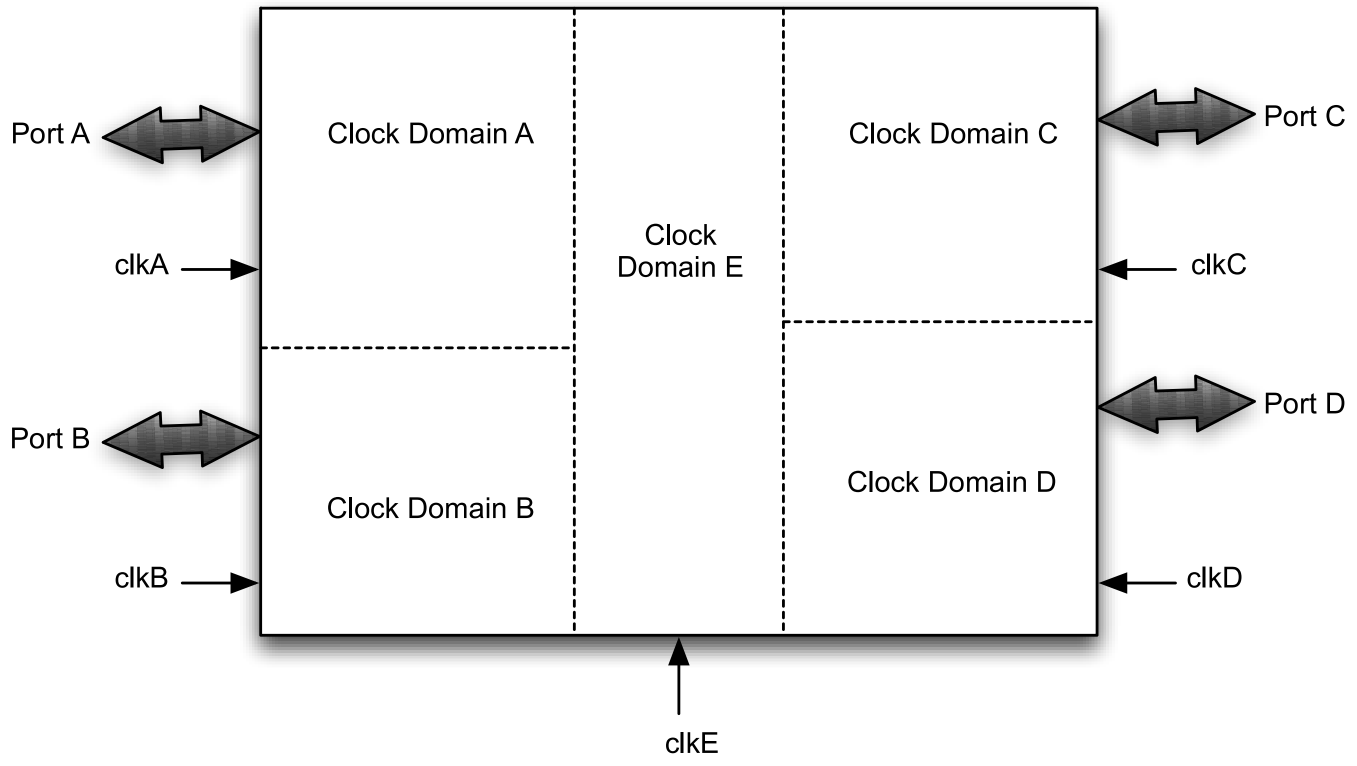
1. Understand what metastability is, and how it can cause failure
2. Understand why metastability happens
3. Be able to design a circuit to reduce the probability that metastability causes system failure
4. To be able to calculate the mean time between failure due to metastability
5. To understand how to use two flip-flops as “synchronizers” to mitigate metastability in single-bit signals
6. To understand how FIFOs can be used to synchronize signals in a multi-clock domain circuit

Multiple Clock Domains

In all our examples so far, the entire circuit is clocked on a single clock

In reality, there are often multiple clock domains

Eg: a network switch

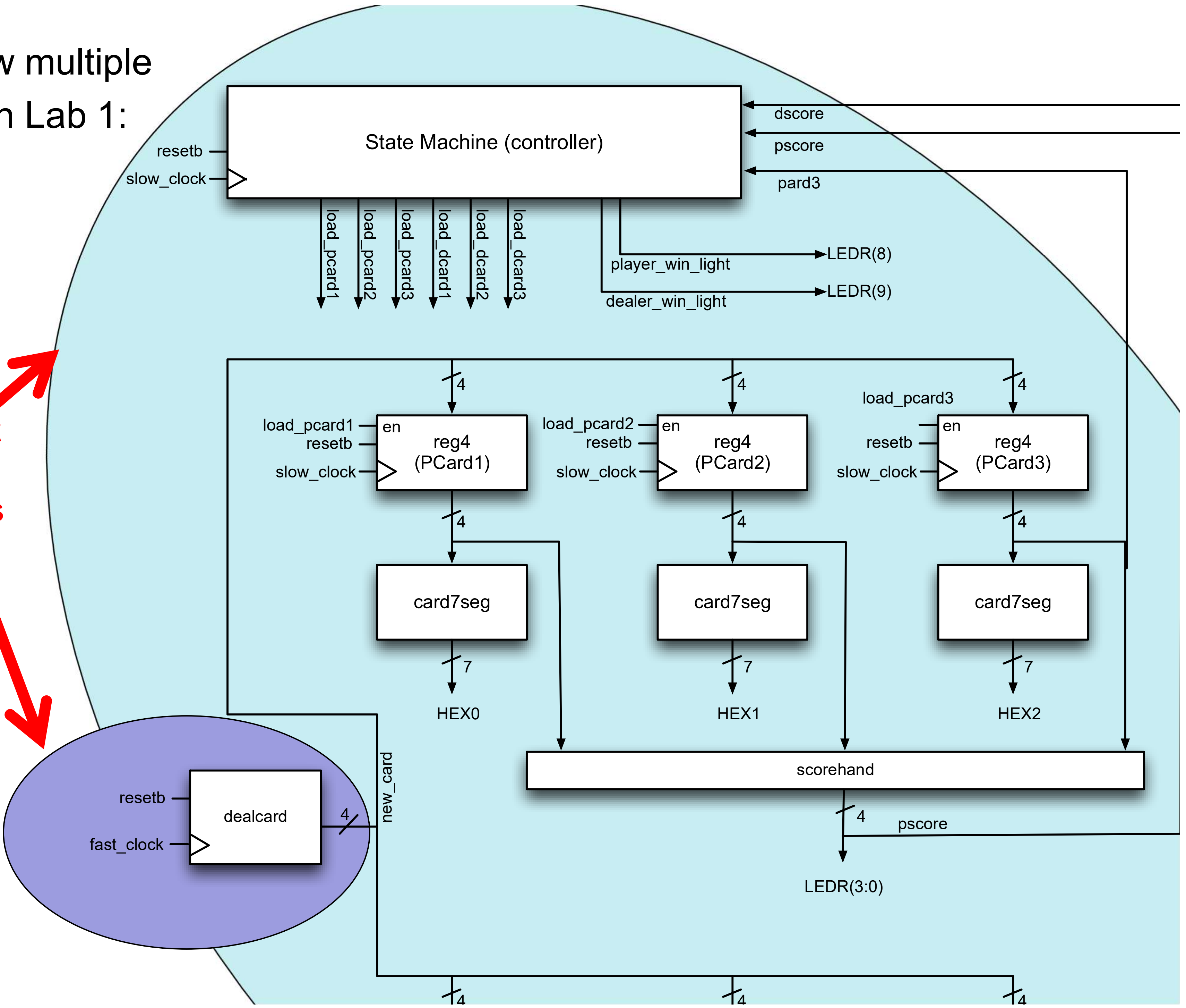


All flip-flops in each clock domain are clocked by the same clock

The clocks may be multiples of each other, or they may be entirely independent

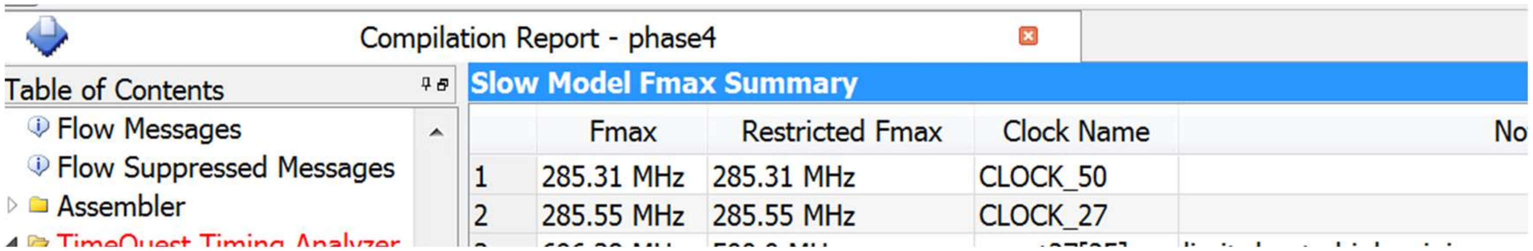
You saw multiple clocks in Lab 1:

Different Clock Domains



Easy to use >1 clock in any circuit

Clock domains identified automatically by FF 'clock' connections:



Compilation Report - phase4

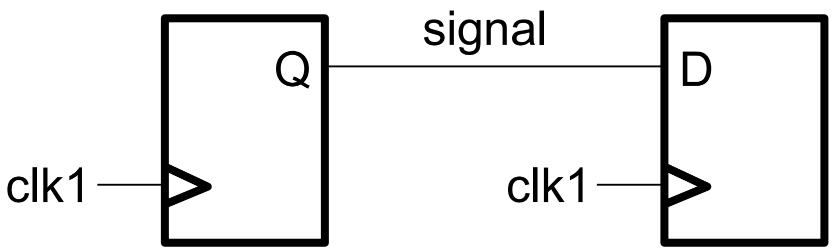
Table of Contents

- Flow Messages
- Flow Suppressed Messages
- Assembler
- TimeQuest Timing Analyzer

Slow Model Fmax Summary

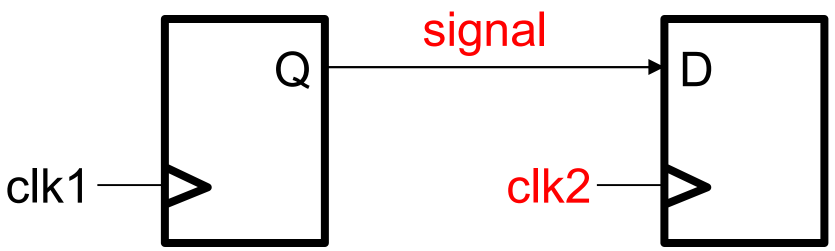
	Fmax	Restricted Fmax	Clock Name	No
1	285.31 MHz	285.31 MHz	CLOCK_50	
2	285.55 MHz	285.55 MHz	CLOCK_27	

- Signals within same clock domain



Easy, same as before

- Signals crossing clock domains…



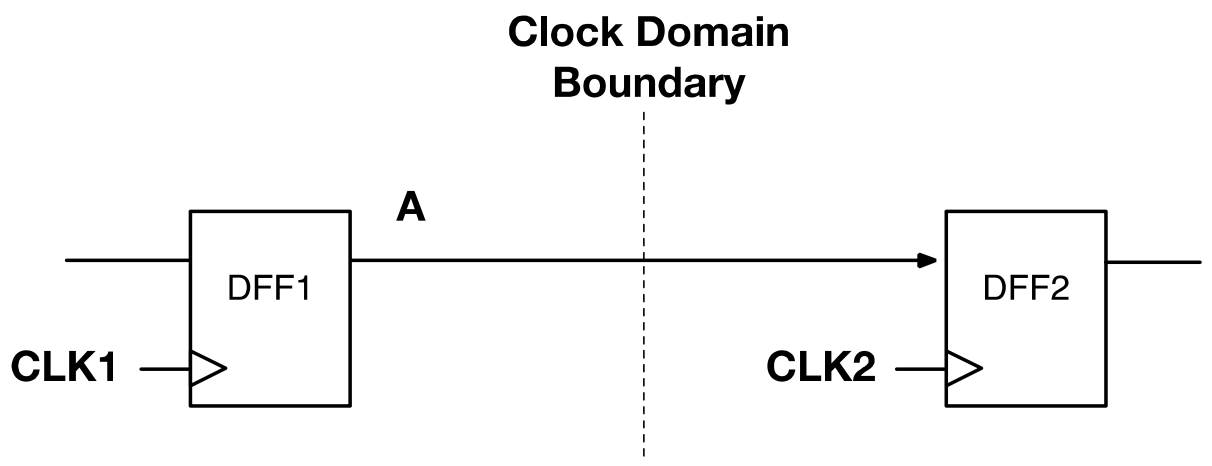
TAKE GREAT CARE HERE!

Asynchronous Signals

- Synchronous signals
  - Launch and capture events use “the same” clock
- “The same”
  - Ideally, same clock source, no clock skew (arrives at same time)
  - Practically, launch clock and capture clock have tightly controlled tolerances
    - Eg, known phase delay, known limit on jitter
    - Can be two different clock sources, as long as they are “phase locked”
  - Controlled tolerances ➔ timing analysis ➔ guarantee setup, hold time conditions
- Asynchronous signals
  - Any signal that isn’t synchronous
  - Includes use of two similar clocks (same clock speed) that are not controlled (synchronized to each other)
  - Includes when launch clock is unknown or not present
    - Many real-life events (eg, human pushing button) are async
- Asynchronous signals are ***The Problem***
  - Leads to metastability

Clock Domain Crossings

Some signals will cross clock domains.

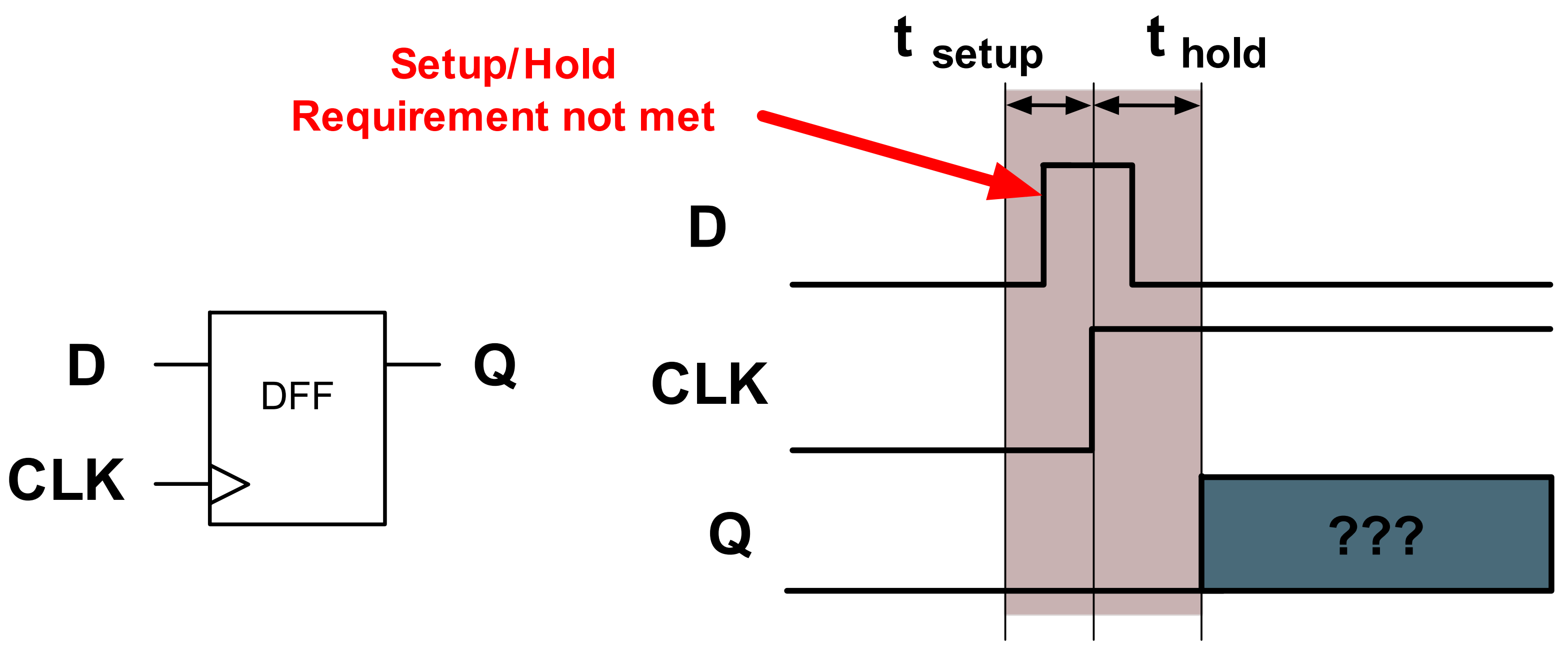


Signal A appears asynchronous to DFF2

- Senders using clk1 ➔ synchronous (same as before)
- Sender (DFF1) is not using clk2 ➔ asynchronous (new)

This is a problem. Why?

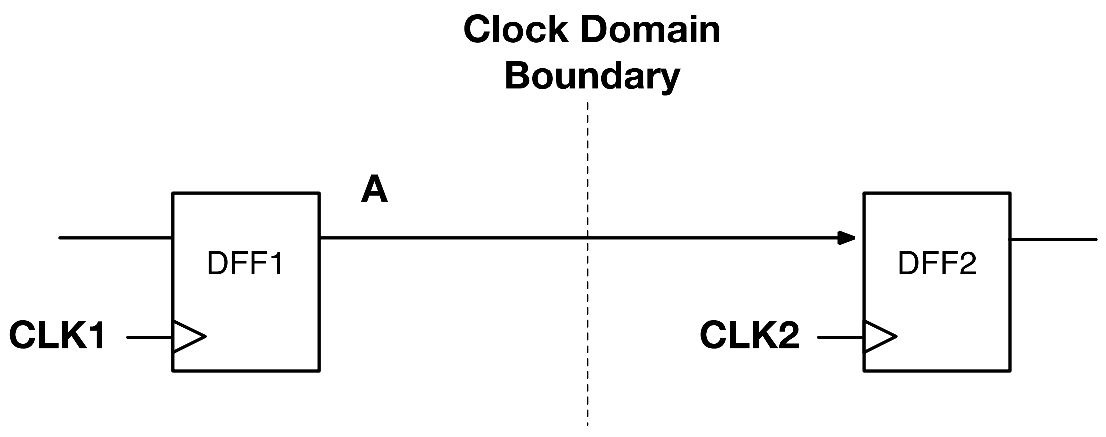
“Forbidden Zone” when the D input cannot change



If the clk1 (sender) and clk2 (receiver) are truly asynchronous

➔ D changes at any time (relative to clk)

➔ D can change in Forbidden Zone



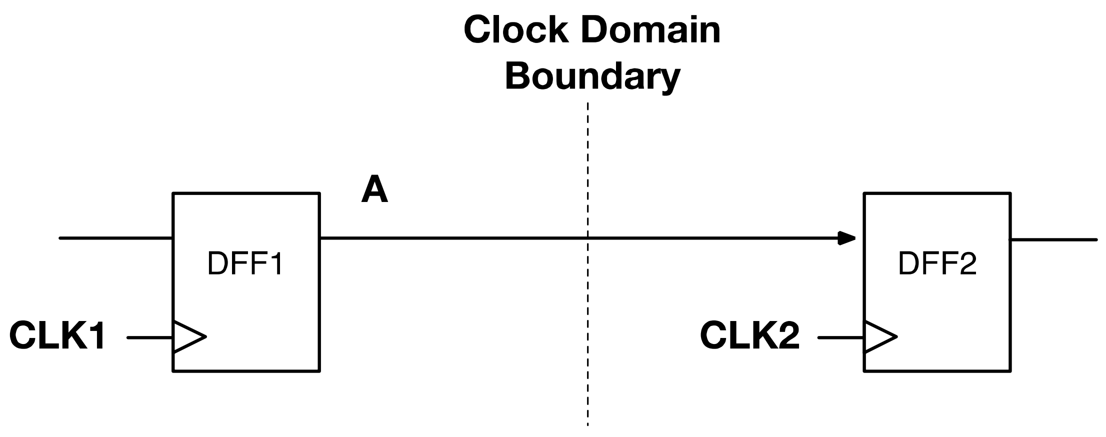
Signal A appears asynchronous to DFF2

- Period of CLK2 = t_{clk} = 1/freq(clk2)
- Forbidden zone length = $t_{setup} + t_{hold}$

Probability A changes in Forbidden Zone (each cycle of clk2):

$$= (t_{setup} + t_{hold}) / t_{clk}$$
$$= \text{freq}(clk2) * (t_{setup} + t_{hold})$$

- Higher probability when…
  - setup/hold times are large
  - clk2 is high frequency



Signal A appears asynchronous to DFF2

Probability A changes in Forbidden Zone (each cycle of clk2):

$$= (t_{setup} + t_{hold}) / t_{clk}$$
$$= freq(clk2) * (t_{setup} + t_{hold})$$

But signal A changes each clk1 (up to clk1 times per second)

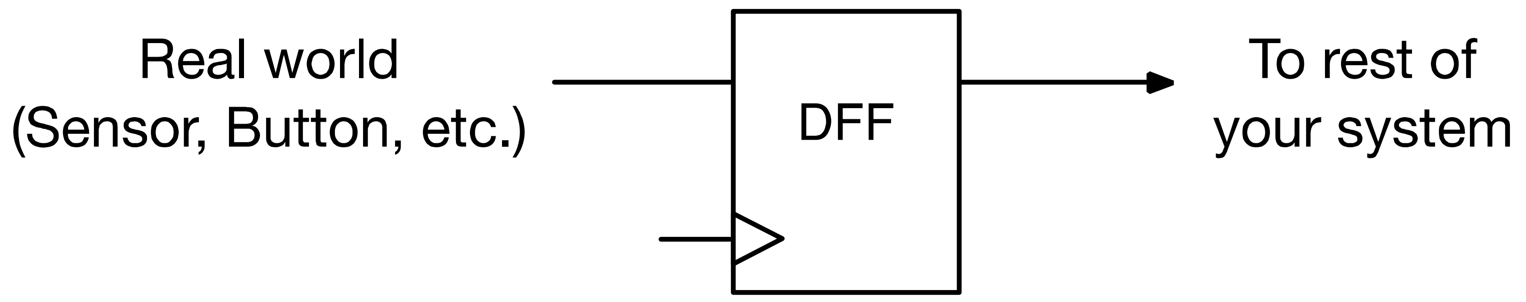
Expected number of violations per second:

$$= freq(clk1) * freq(clk2) * (t_{setup} + t_{hold})$$

Asynchronous Signals

A related problem: Real World Signals

- Real world signals are asynchronous
  - Eg, buttons aren't aligned to clk
- Signals can change too close to clock edge



Probability of violations?

- Same analysis as previous slides
- Use expected number of transitions per second of the real world signal

Important point from previous slides:

In a system with one clock, we can adjust the clock to ensure we never violate the forbidden zone

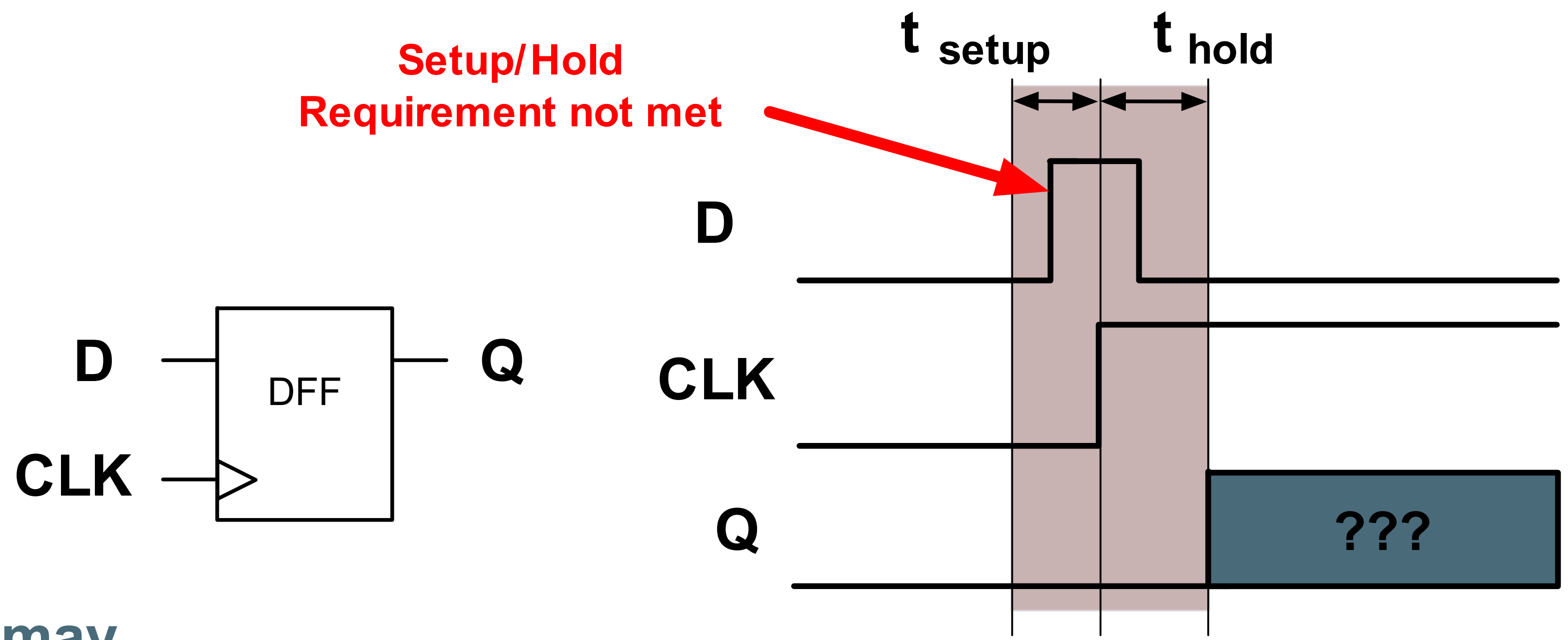
In a system with multiple independent clocks or asynchronous inputs, we can't

- and, violations are going to happen often…

So we better talk about what happens if we change the flip-flop in the…

What happens if we violate a flip-flop’s setup or hold time requirements?

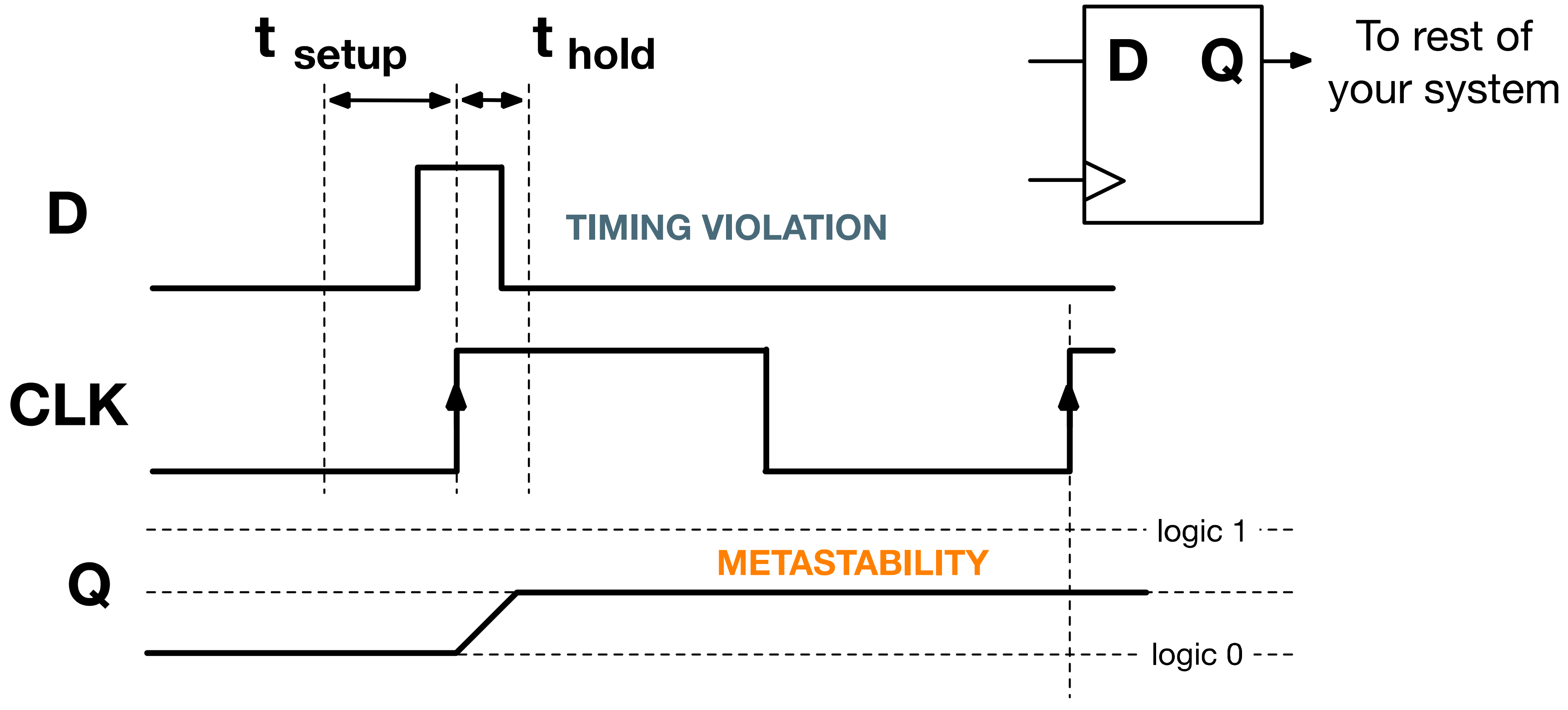
Violating Flip-Flop Setup/Hold Times



Q may…

- get the wrong value (0 or 1)
- get the correct value
- become **metastable**
  - analog value between 0 and 1
    - remember: transistor voltages → continuous range of values
  - or very fast oscillations between 0 and 1 (and 0 and 1 …)

Metastability

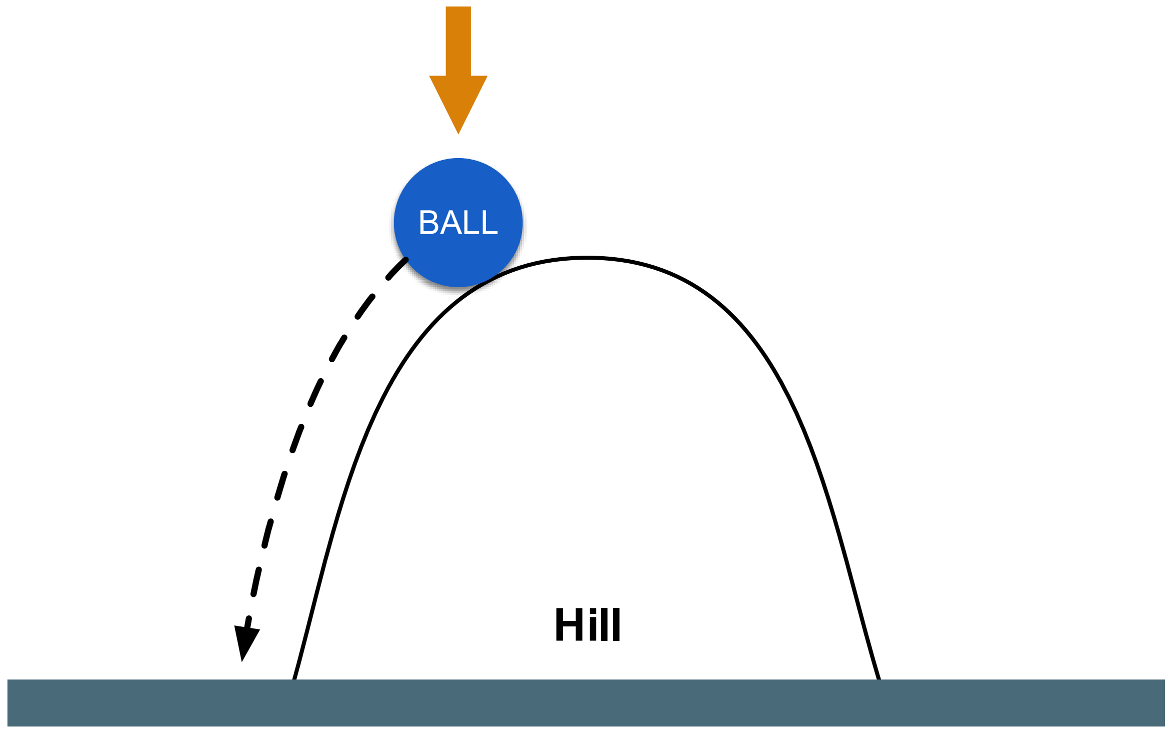


METASTABLE VALUE PROPOGATED TO OTHER PARTS OF SYSTEM

MAY CAUSE SYSTEM-WIDE FAILURE

Why does metastability happen?

Mechanical Analogy



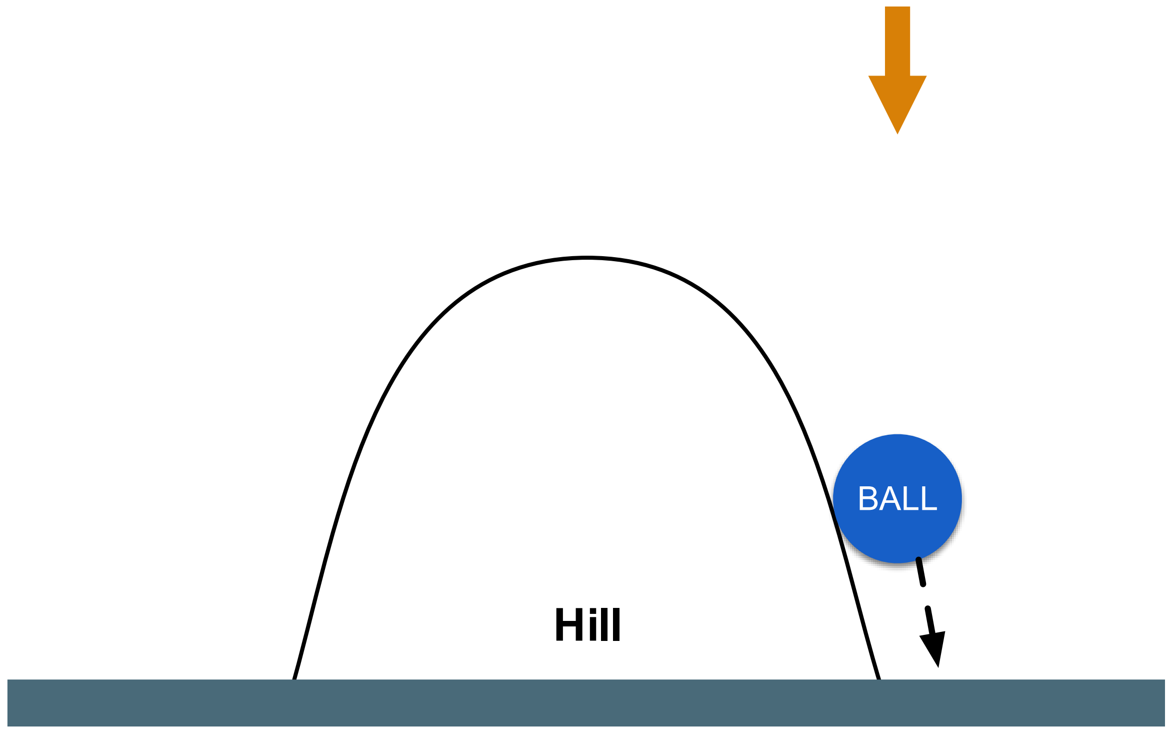
Imagine dropping a ball on a hill

- perfectly smooth and symmetrical hill
- no forces except gravity

Depending on where you drop the ball, it will settle in one of two states

- either on the left side or the right side

Mechanical Analogy

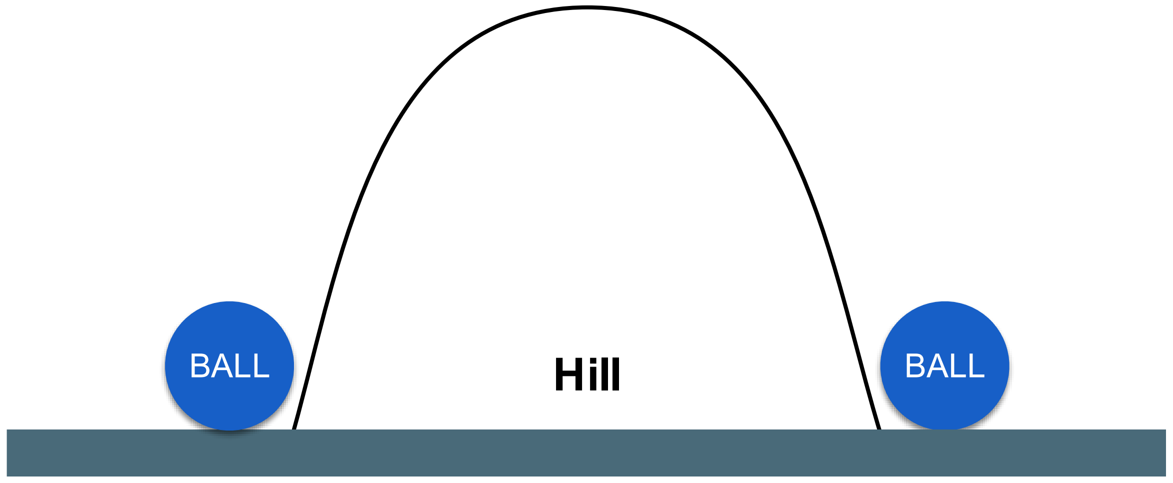


The closer you drop the ball near either side, the faster it settles

- Shorter distance
- Steeper slope

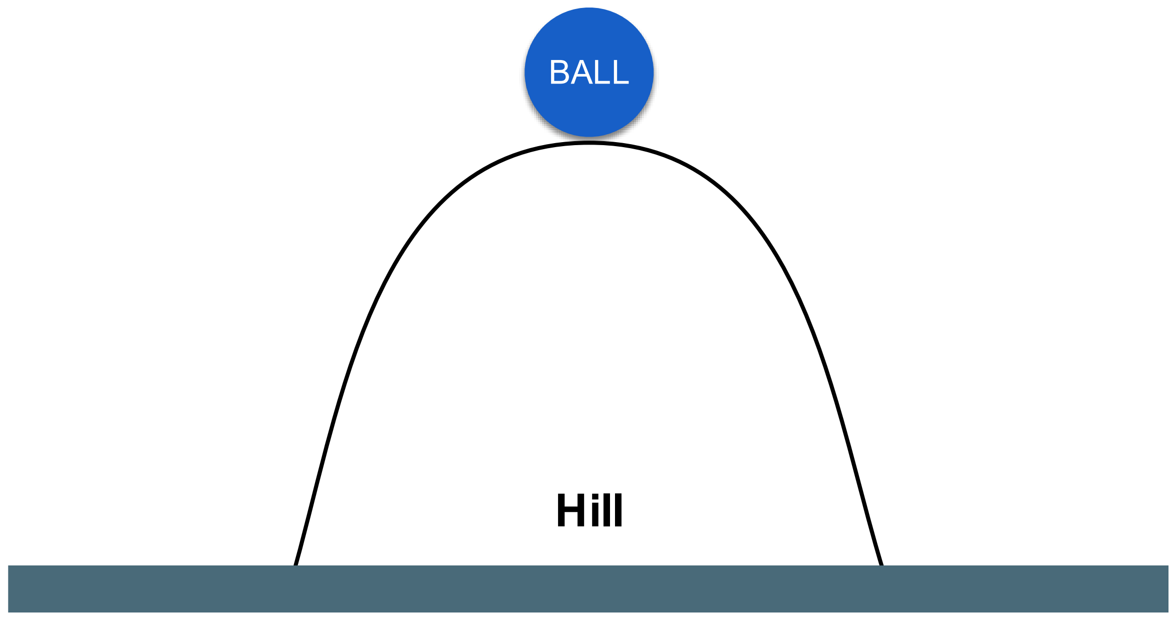
Mechanical Analogy

Two Stable States



Once ball reaches either state, it will stay there forever

There is a THIRD “metastable” state!



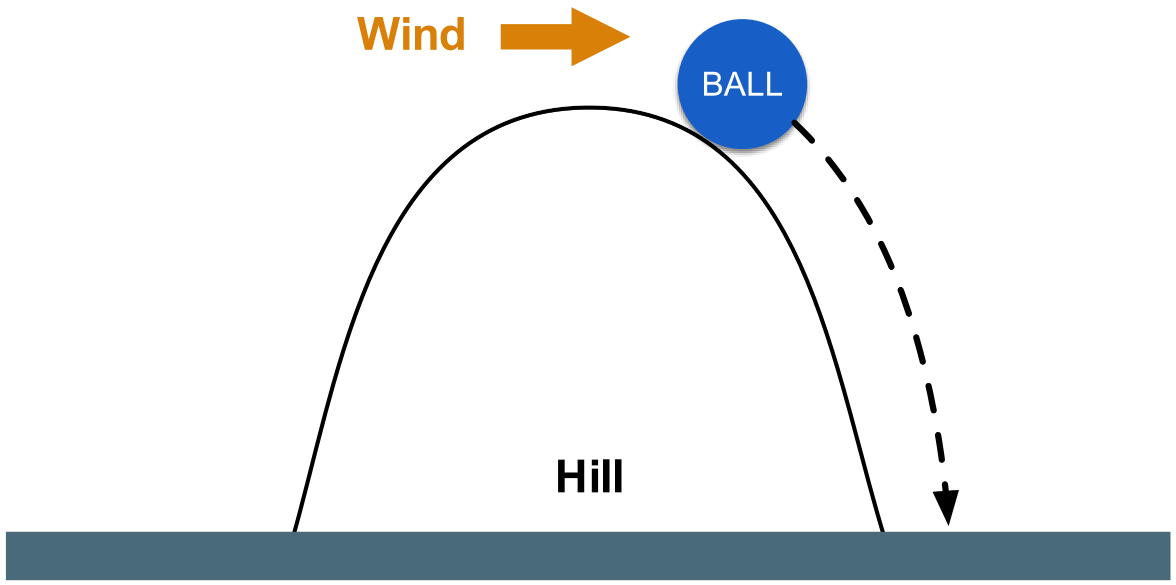
Imagine the ball is dropped perfectly in the middle

The slope at the exact middle is flat

There are no other forces besides gravity

In theory, ball will stay there forever

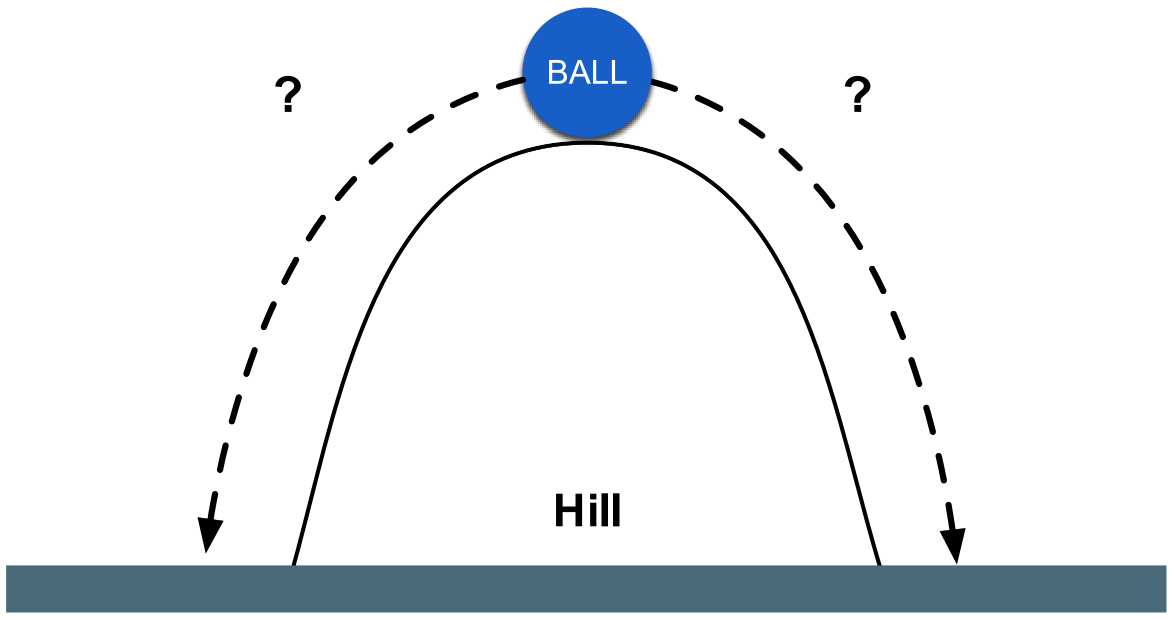
Mechanical Analogy



In practice, ball will eventually settle to a stable state

- Gust of wind
- Non-perfect hill
    - Not perfectly symmetrical
    - Slope not perfectly flat in the middle

Mechanical Analogy



Into which state will it settle?

How long before it settles?

Not predictable!

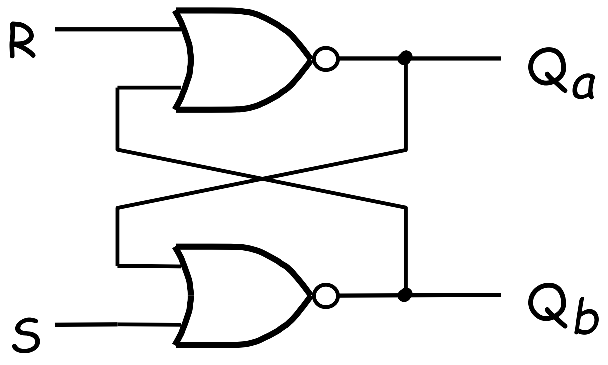
OK. But how does this apply to flip-flops?

Let’s first review what a DFF looks like inside

S-R Latch

We will only operate this under the following conditions:

1. Qa and Qb are always inverse of each other
2. R and S inputs are never 1 at the same time



If R=1, S=0, Qa=0 and Qb=1 ("reset")

If R=0, S=1, Qa=1 and Qb=0 ("set")

If R=0, S=0, Qa and Qb maintain value

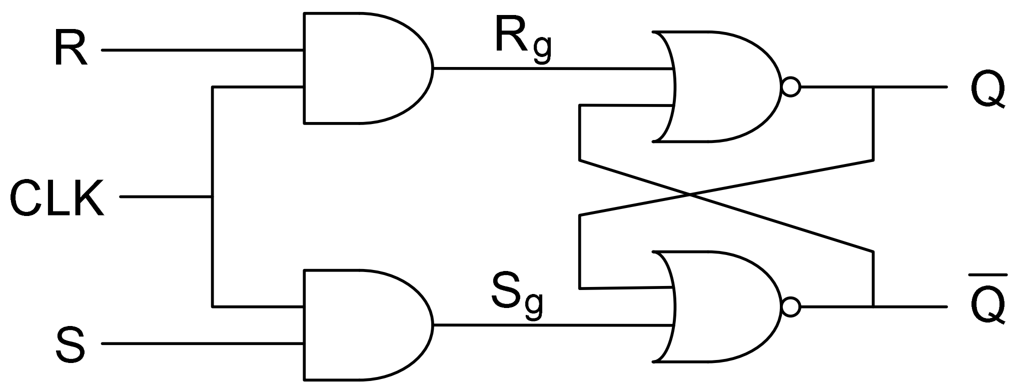
Important point: in an S-R latch:

- If the set signal goes high, the output Qa goes to 1
- If the reset signal goes high, the output Qa goes to 0
- If neither set nor reset are high, the output <u>maintains its value</u>

What about if S and R are high at the same time?

- *Would not normally use an S-R latch this way*

Gated S-R Latch

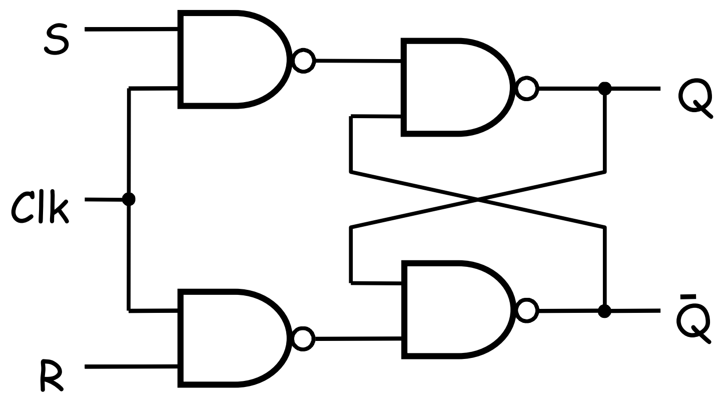


When clk is high, it operates like a normal S-R Latch

When clk is low, the latch maintains value, regardless of R and S

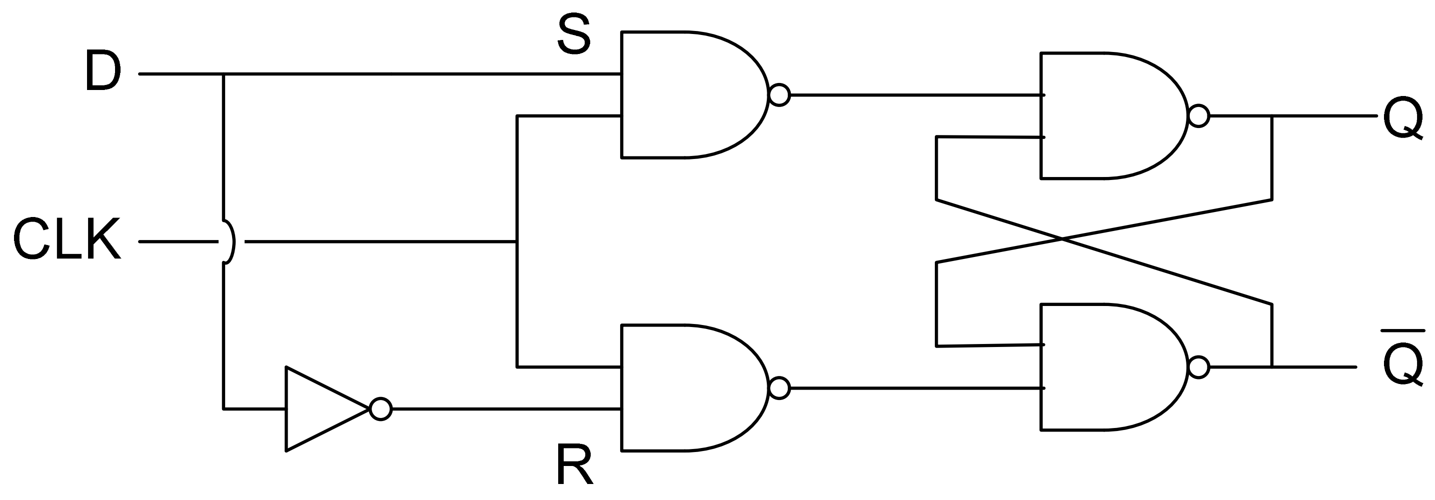
This is called level-sensitive, since the operation depends on the level of the clk signal.

By the way, this has the same function as the Gated S-R latch:



Who cares? It turns out it uses fewer transistors

Gated D Latch



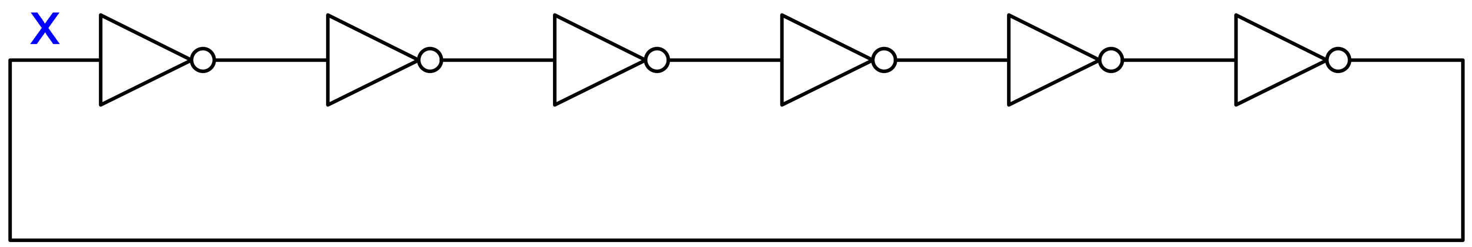
If CLK is high, Q becomes equal to D

If CLK is low, Q maintains its value (regardless of D)

This is <u>level-sensitive</u>

How does metastability happen?

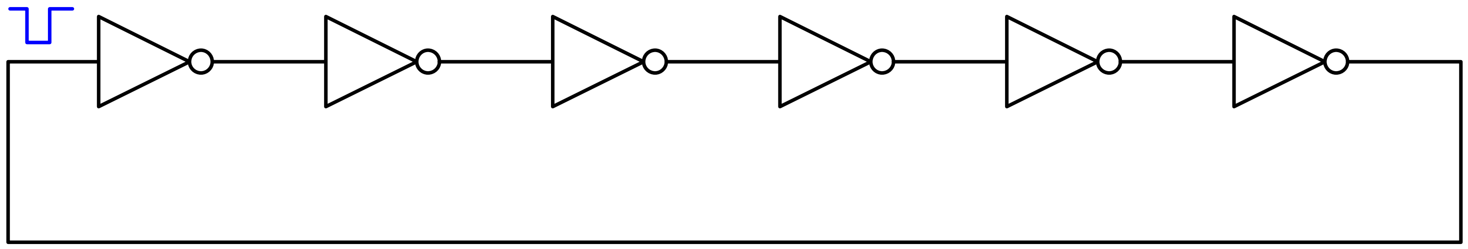
Before answering this question, consider:



Two steady states: x=0 or x=1

How does metastability happen?

Now put in a short pulse at x (about one gate delay long)



Pulse races around ring, causing oscillation

-Theoretically, it could oscillate forever

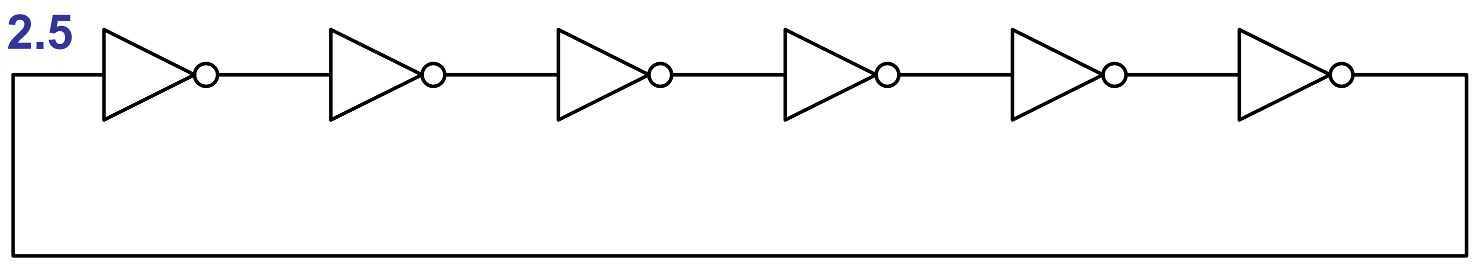
-Reality: eventually, the pulse will get narrower and narrower

- After some time, it is shrinks to nothing and stops oscillating

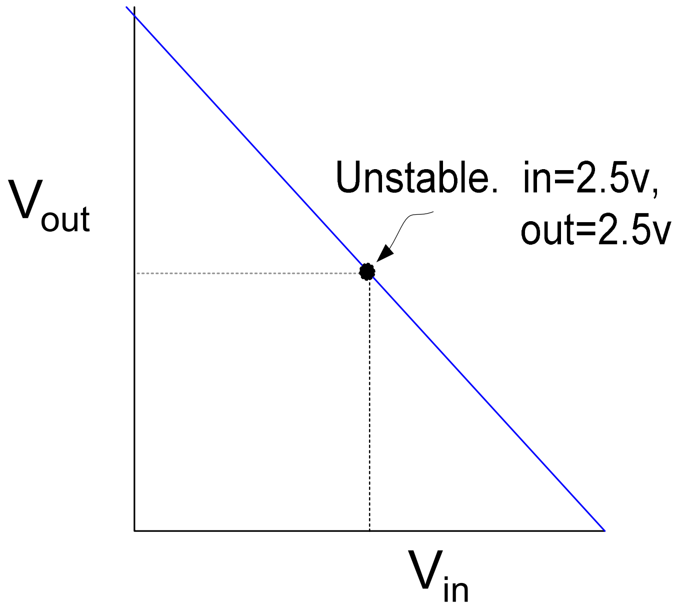
-NB: the pulse could also get wider and wider, but the effect is the same

How does metastability happen?

Now consider this: what if we put in 2.5 volts (assume a 5 volt system?)

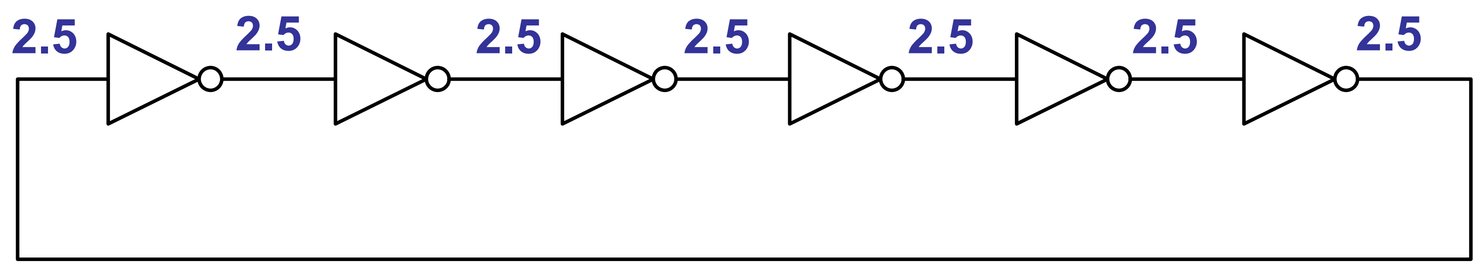


Remember the transfer function of an inverter:



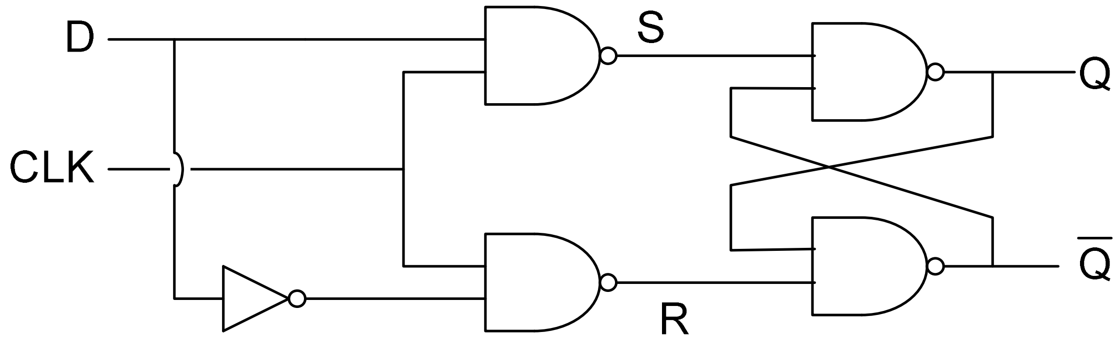
How does metastability happen?

So ideally, we might get:

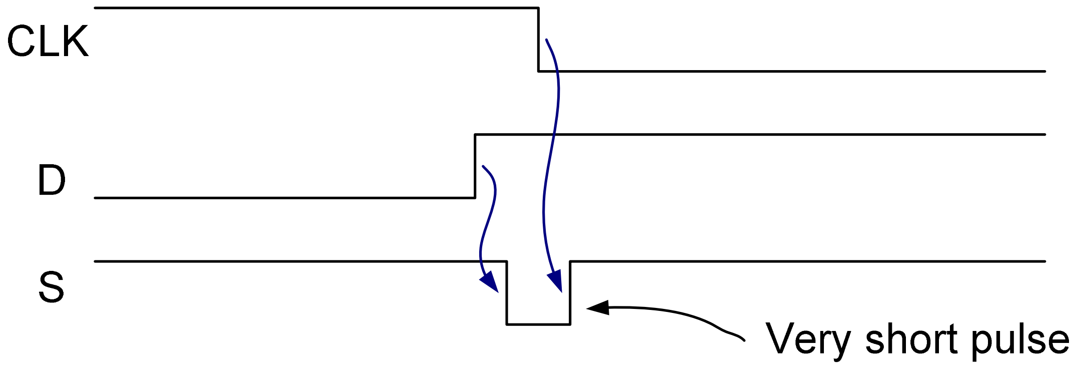


Of course, eventually noise will knock it one way or the other

Now go back to a flip-flop:



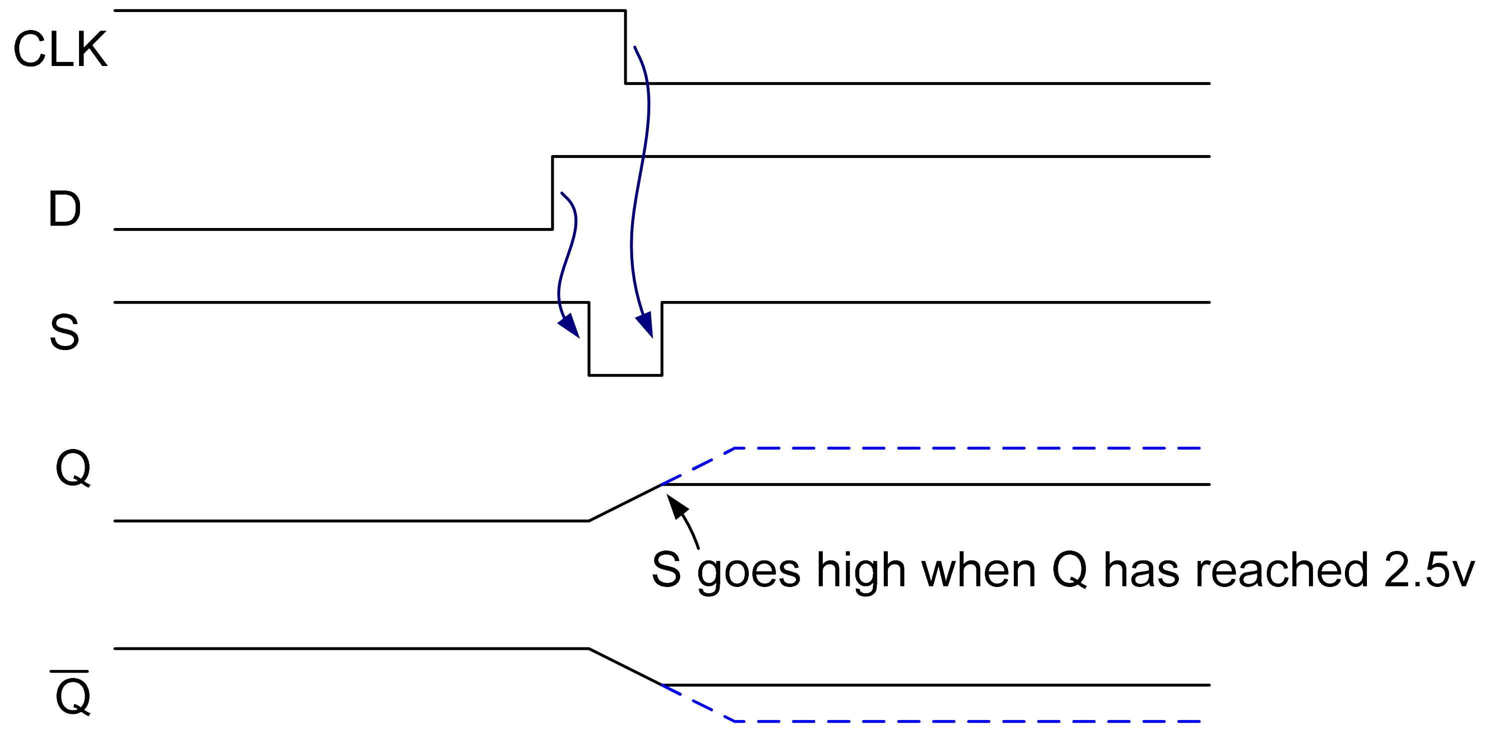
Case 1: If D changes close to CLK:



This short pulse races around the ring of two cross-coupled NAND gates, causing oscillation

→ Output oscillates forever (theoretically)

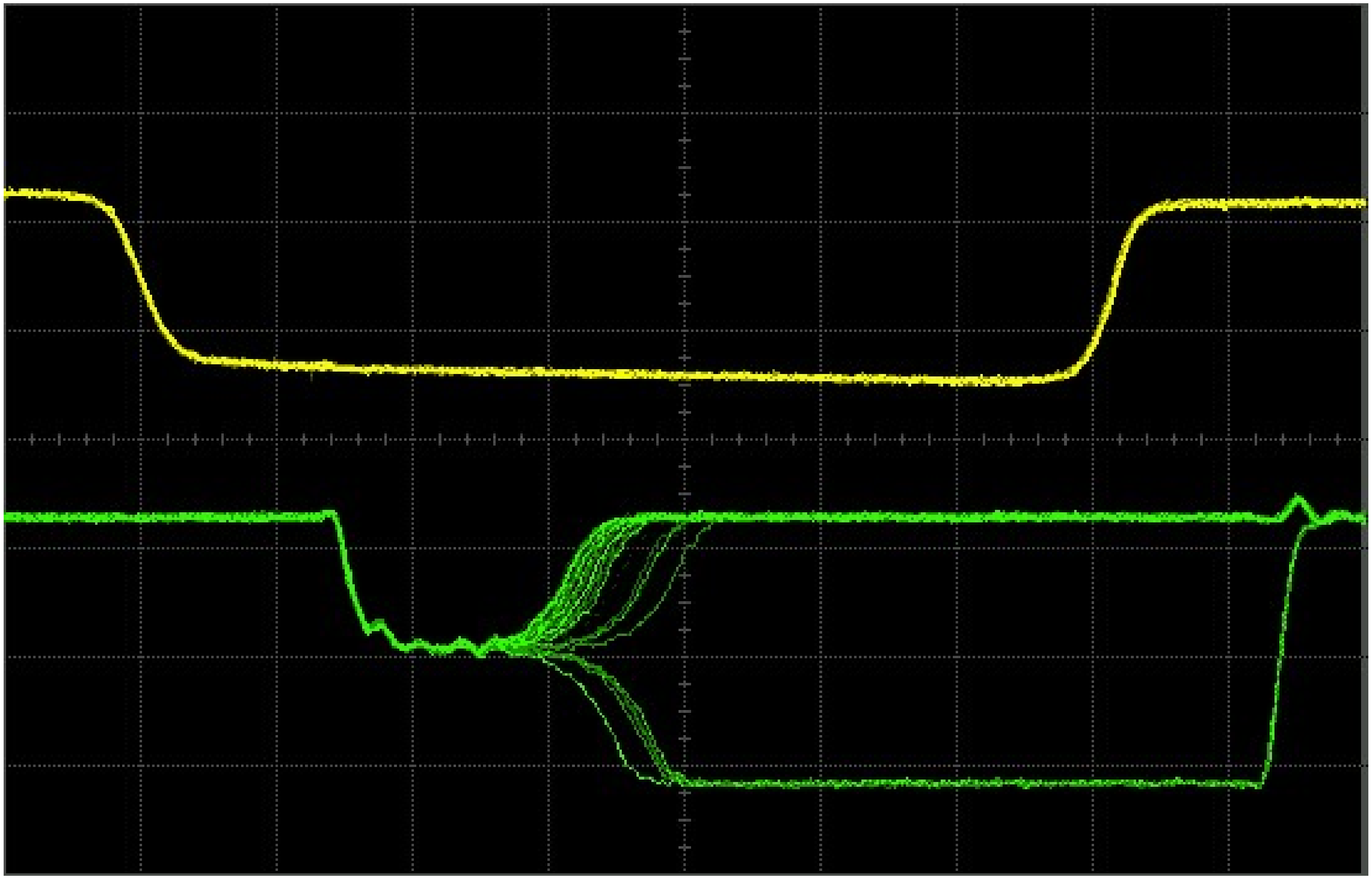
Case 2: If rise time of Q is very slow:



If timing is exactly “right”, Q and Q-bar could be left at 2.5 volts

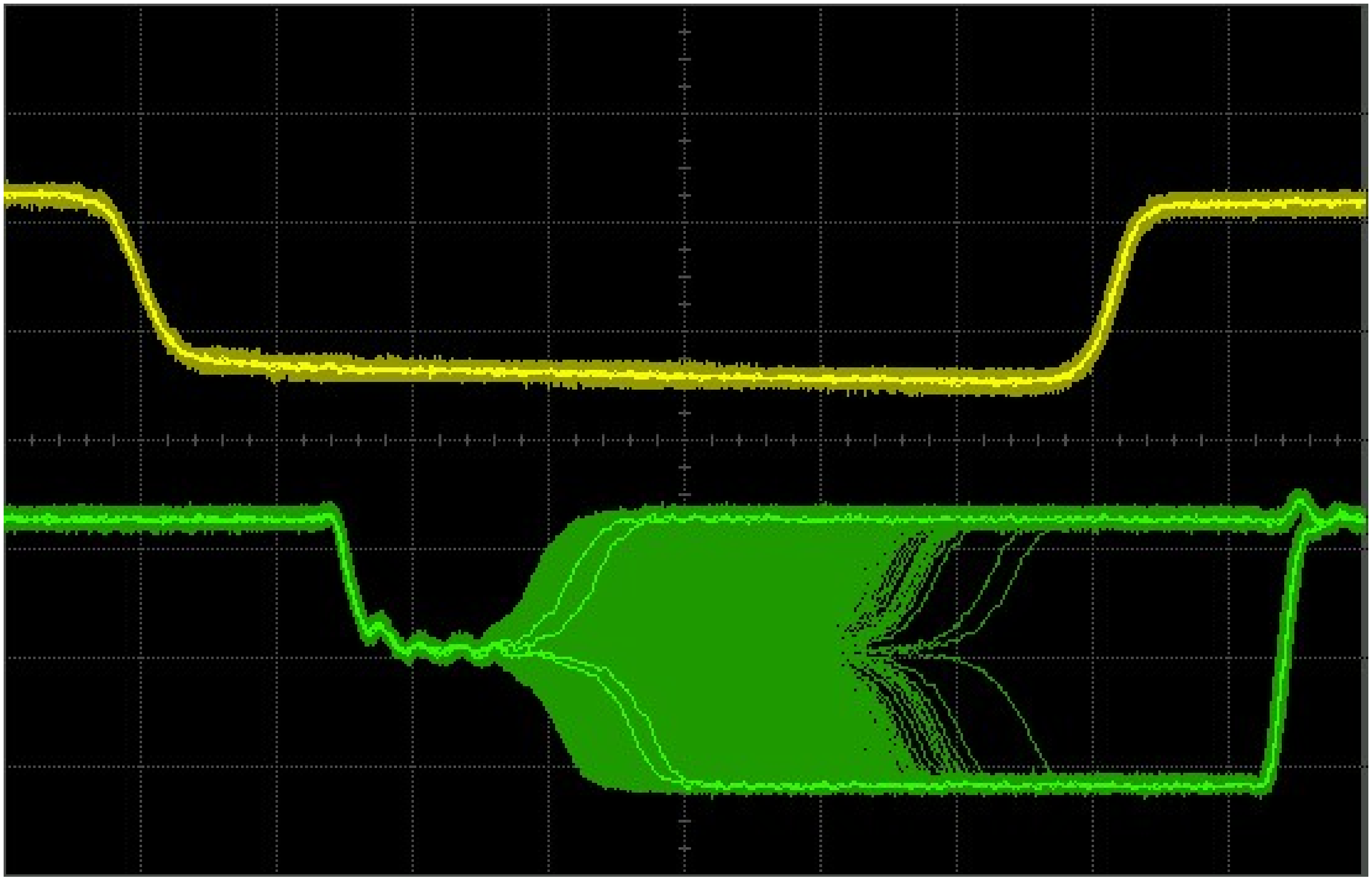
→ Output would remain at 2.5 volts (theoretically forever!)

Real Metastable Example



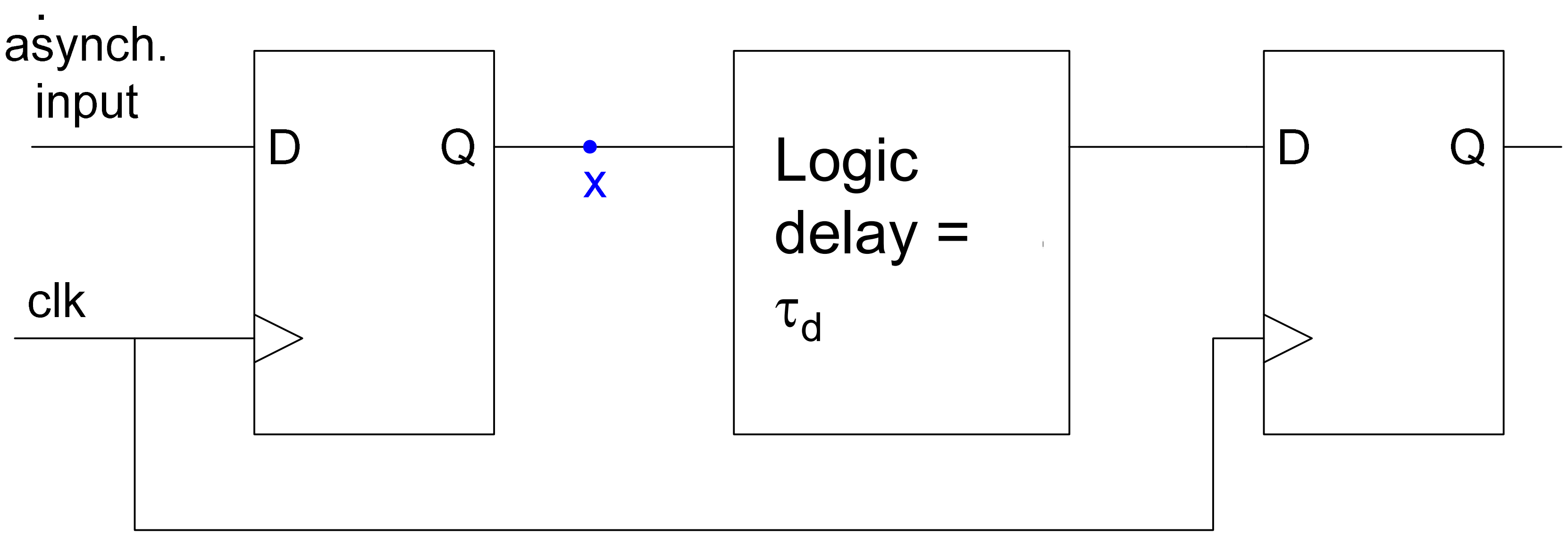
[Dally]

Real Metastable Example



[Dally]

How can metastability cause failure?

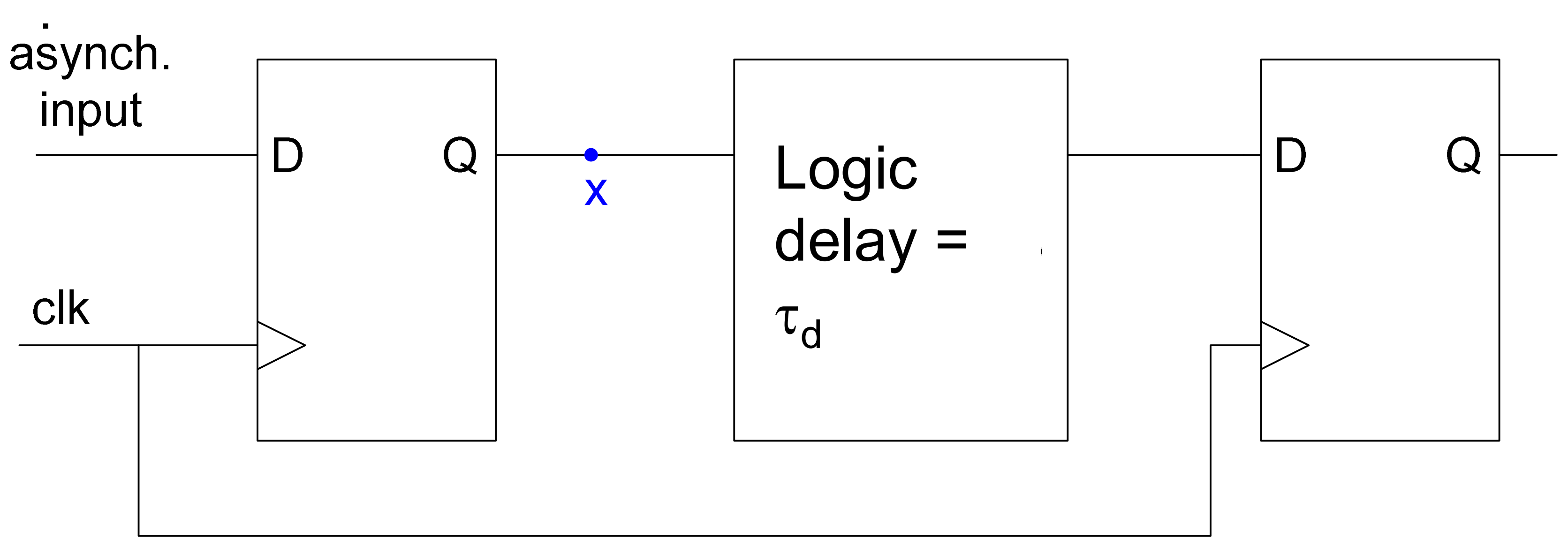


Definition:

$$t_{slack} = \tau_p - (t_{clk\text{-}to\text{-}q} + \tau_d + t_{setup})$$

- Slack: measure of how “close” path timing is to being “critical path”
  - ie, amount of “extra delay” needed to make path critical

How can metastability cause failure?



Suppose node X goes metastable…

- time until node x stops oscillating or settles (0 or 1) is τ_m

If $\tau_m > t_{slack}$ then **system failure!**

Important point:

more slack $\Rightarrow$

more time for metastability to resolve

Mean Time Between Failure (MTBF)

Metric to estimate the average time between two failure-causing instances of metastability on a given signal transfer

$$MTBF\left(t_{slack}\right) = \frac{e^{t_{slack}/C_0}}{C_1 \cdot f_{CLK} \cdot f_{DATA}}$$

- f_{CLK} – Clock frequency
- f_{DATA} – Toggling frequency of the FF input
- t_{slack} – Amount of slack available on the path
- C_o, C_1 – Constants dependent on operating conditions and process tech

Mean Time Between Failure (MTBF)

Metric to estimate the average time between two failure-causing instances of metastability on a given signal transfer

$$MTBF\left(t_{slack}\right) = \frac{e^{t_{slack}/C_0}}{C_1 \cdot f_{CLK} \cdot f_{DATA}}$$

Higher MTBF ➔ more robust design

- maybe hundreds or thousands of years

Required MTBF depends on application

- Life-critical medical equipment → higher MTBF
- Consumer video game → lower MTBF

MTBF Example Calculation - Synchronizer

Example without much slack:

$$f_{CLK} = 100\text{MHz}$$
$$f_{DATA} = 1\text{MHz}$$
$$t_{slack} = 0.5\text{ns}$$
$$C_o = 0.31\text{ns}$$
$$C_1 = 9.6 * 10^{-18}\ \text{s}$$

MTBF = 1.53 hours

Example with more slack:

$$f_{CLK} = 100\text{MHz}$$
$$f_{DATA} = 1\text{MHz}$$
$$t_{slack} = 3\text{ns}$$
$$C_o = 0.31\text{ns}$$
$$C_1 = 9.6 * 10^{-18}\ \text{s}$$

MTBF = 12 years

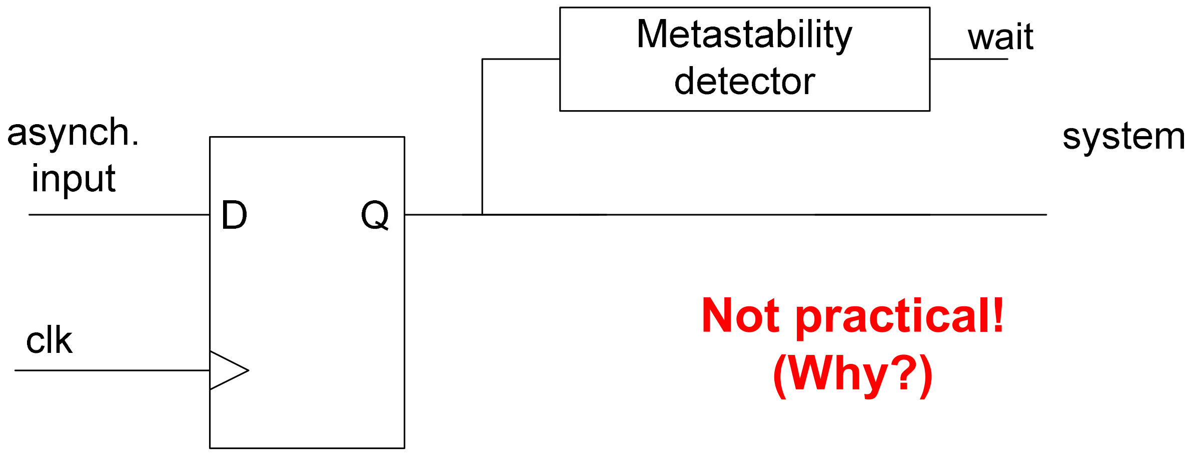
Note: MTBF is very sensitive to slack

Dealing with Metastability:

Possible options:

1. Eliminate possibility of metastability
    - Use only asynchronous circuits (usually not feasible)
    - (future courses)

2. Handle metastability when it occurs

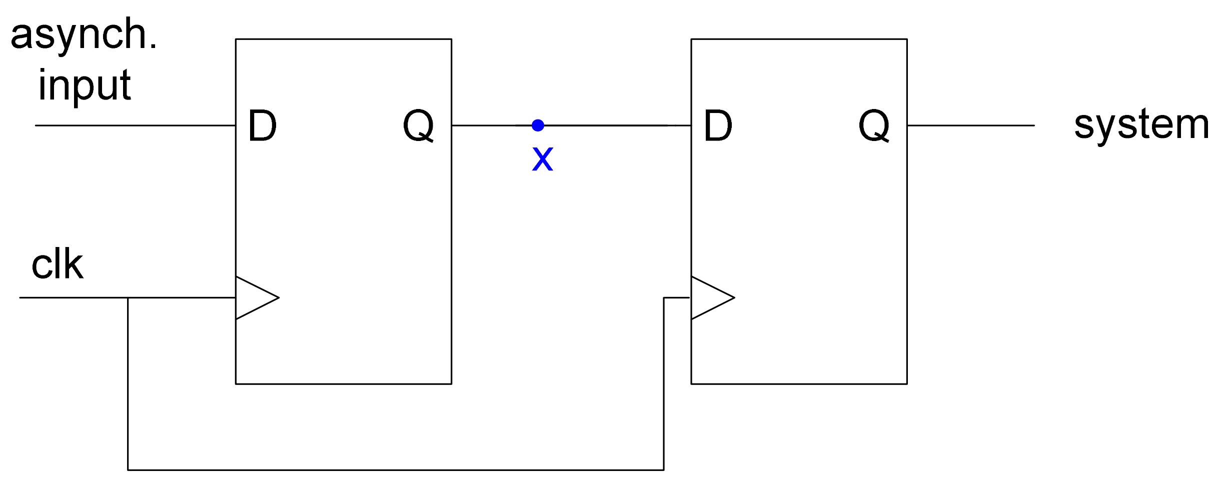


Not practical!
(Why?)

- If Q goes metastable, “wait” signal causes system to wait until metastability resolves itself

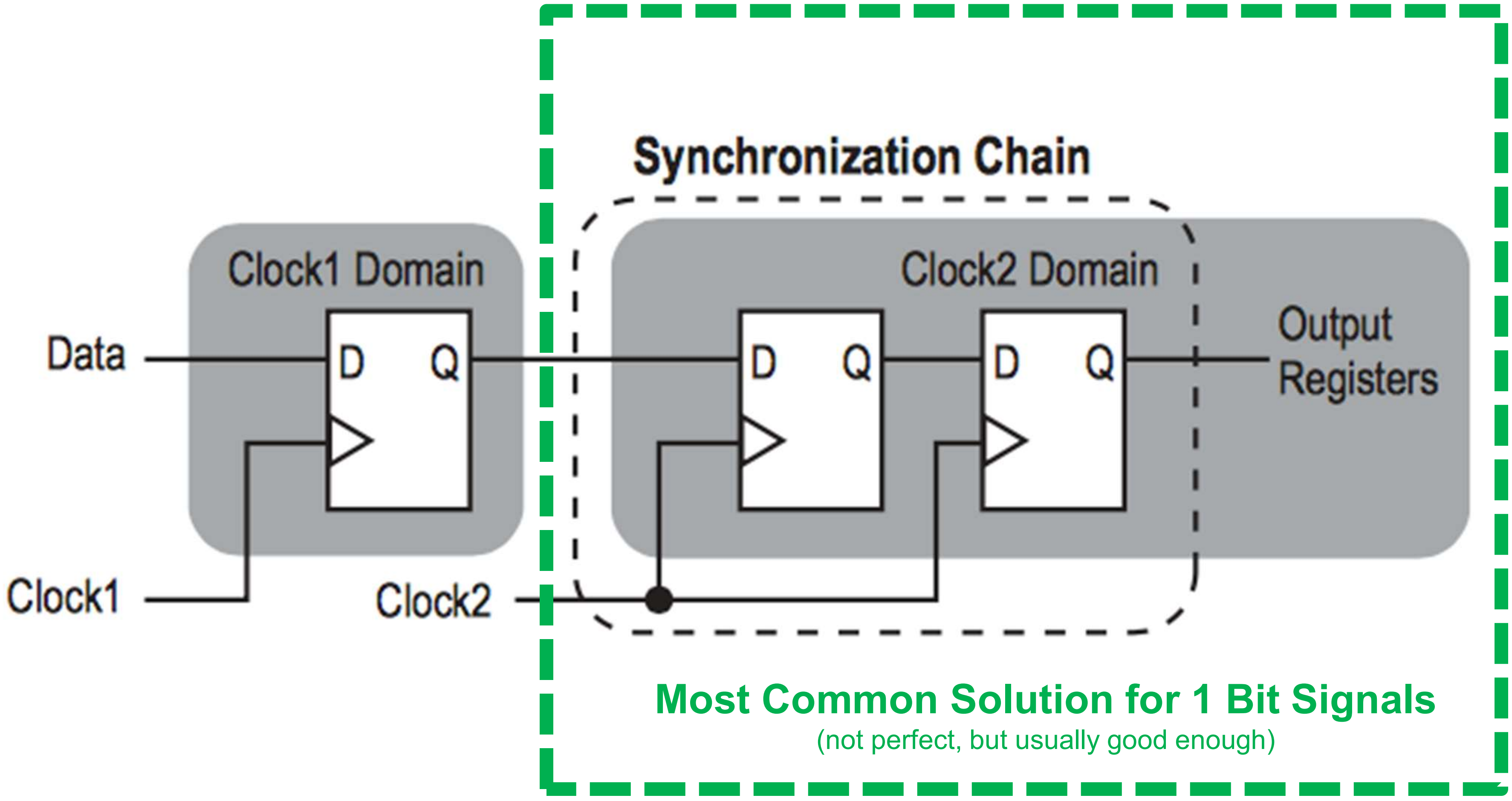
Dealing with Metastability:

3. Reduce the probability that metastability would cause system failure:



Double-Register each asynchronous input

- If X goes metastable, it has an entire clock cycle to resolve itself
- This technique is very common

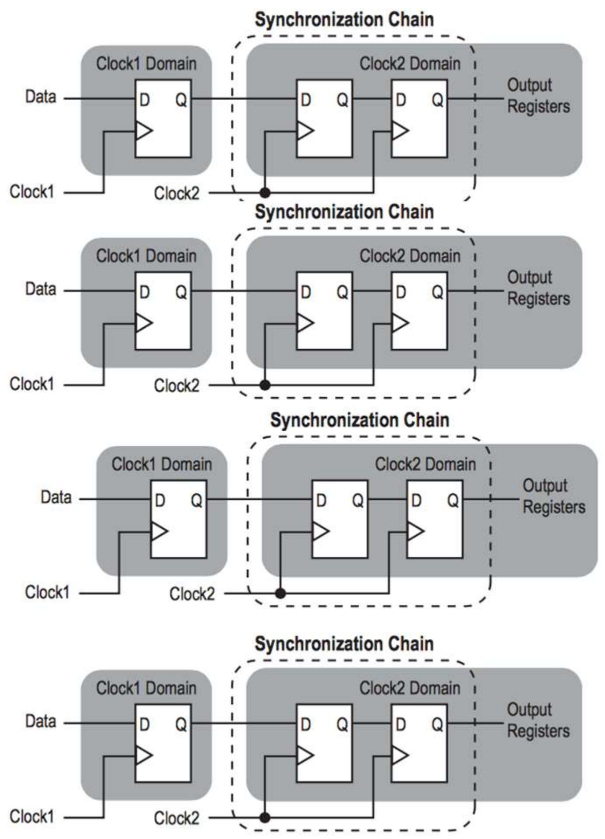


Most Common Solution for 1 Bit Signals

(not perfect, but usually good enough)

Transmitting Multi-bit Words

Transmitting multi-bit words (buses) between timing domains is also troubling. Suppose we have a four bit word. We could use one synchronizer on each bit:



Suppose each bit has a slightly different delay. In this case, if the change is close to the receiving clock edge, some bits may take their old value and some may take their new value

Previous 4b data item: 0 1 1 0

Next 4b data item: 1 0 1 1

Suppose the MSB is slower than the others

- eg, MSB arrives just after the receiving clock edge
- receiver sees 0 0 1 1 instead of 1 0 1 1

It would be resolved in the next cycle

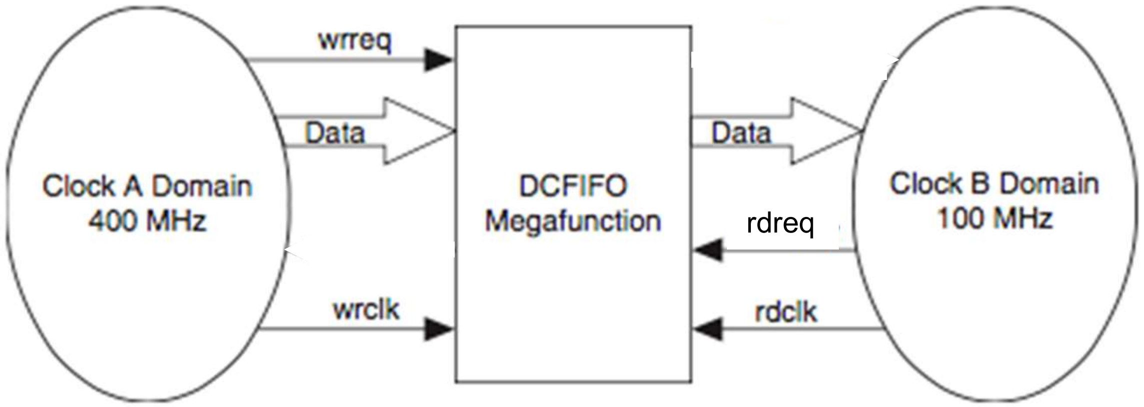
- However, for one cycle, you are operating on an invalid word
- May lead to system failure (if you don't anticipate this)

Solution 1

- Use a “Gray code”
  - Only 1 bit of data can change per cycle
- Not always possible

Solution 2

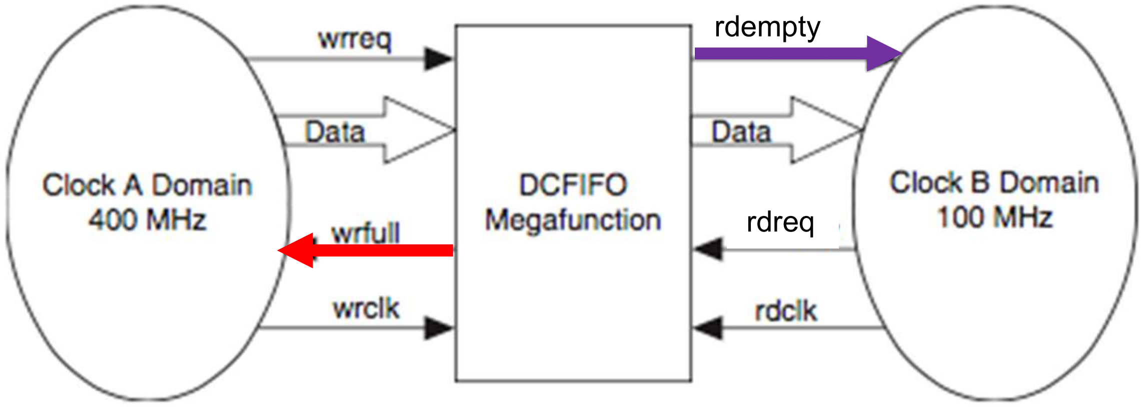
- Use a double-clocked FIFO (DCFIFO)
  - Write side of the FIFO → sender clk
  - Read side of FIFO → receiver clk
- FIFO is specially designed for clock-domain crossing



In this course, we won't design DCFIFO

- Think about it. Google it ☺

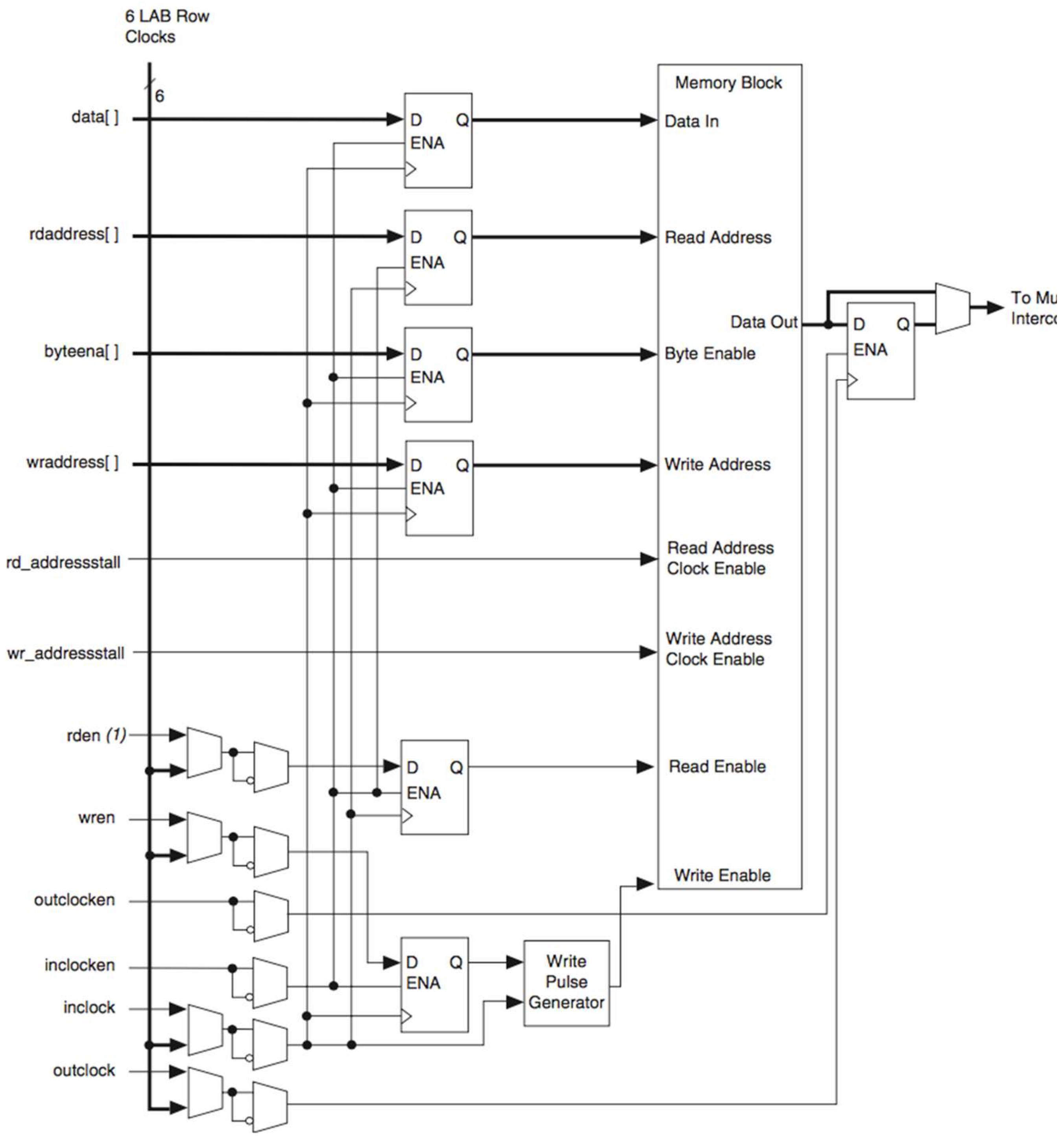
Another advantage of DCFIFO: flow control



Producer / Consumer at different rates

- Flow control using wrfull / rdempty flags

In our device, can implement FIFO using an embedded RAM



From Altera, Cyclone II handbook

Learning Objectives

1. Understand what metastability is, and how it can cause failure
2. Understand why metastability happens
3. Be able to design a circuit to reduce the probability that metastability causes system failure
4. To be able to calculate the mean time between failure due to metastability
5. To understand how to use two flip-flops as “synchronizers” to mitigate metastability in single-bit signals
6. To understand how FIFOs can be used to synchronize signals in a multi-clock domain circuit